Using mixtures of common ancestors for estimating the probabilities of discrete events in biological sequences*

Eleazar Eskin†and **William Noble Grundy**
Department of Computer Science
Columbia University
{eeskin,bgrundy}@cs.columbia.edu

Yoram Singer
School of CSE
Hebrew University
singer@cs.huji.ac.il

Abstract

Accurately estimating probabilities from observations is important for probabilistic-based approaches to problems in computational biology. In this paper we present a biologically-motivated method for estimating probability distributions over discrete alphabets from observations using a mixture model of common ancestors. The method is an extension of substitution matrix-based probability estimation methods. In contrast to previous substitution matrix-based methods, our method has a simple Bayesian interpretation. The method presented in this paper has the advantage over Dirichlet mixtures that it is both effective and simple to compute for large alphabets. The method is applied to estimate amino acid probabilities based on observed counts in an alignment and is shown to perform comparable to previous methods. The method is also applied to estimate probability distributions over protein families and improves protein classification accuracy.

1 Introduction

Many successful approaches to modeling sequences for computational biology problems have involved statistical models. Typically these problems are multi-class classification problems which involve classifying a sequence into one of a set of possible classes. For example, the protein homology problem consists of classifying an unknown protein sequence into a protein family. Traditionally, these models are generative models where a model is trained over one class of data. An unknown protein sequence is evaluated by each protein family model to determine which protein family is most likely to have generated the protein sequence. These models include approaches such as hidden Markov model-based approaches [24, 9, 2, 23], probabilistic suffix tree-based approaches [3, 1] and profiles-based approaches [12]. Recently discriminative models have been applied to protein homology and approaches have included using support vector machines [21] and sparse Markov transducer approach [10]. In discriminative models, the model is trained over data containing multiple classes. These discriminative approaches directly classify an unknown protein sequence into a family.

Many statistical models employ a mechanism for estimating a probability distribution over a given alphabet from a set of observations of that alphabet. For generative protein homology approaches such as hidden Markov models or profiles, this mechanism is used for estimating the probability of observing an amino acid in a certain portion of the model given a set of observations of amino acids at that position. Analogously, for some discriminative statistical models such as sparse Markov transducers, this mechanism is used for estimating the probability over protein families in a certain portion of the model given a set of observations of protein families. For estimating over amino acids, many approaches have been presented (see below). These approaches can be applied to other alphabets such as protein families for use in discriminative models. However, one problem is that the size of the alphabet is for protein families is significantly larger

*Keywords: multinomial estimation, regularizers, amino acids, protein families

†Phone Number: (212) 939-7078 Fax Number: (435) 408-3083 Postal Address: 450 CS Building/Department of Computer Science/500 W. 120th St./New York, NY 10027

($\approx 2,500$) than the size of the alphabet amino acids (≈ 20). Because of the large alphabet, it is difficult to apply some of the best performing approaches developed for amino acids. In this paper, we present an new efficient robust method to compute these probability distributions which performs well on large alphabets. The method uses a mixture of common ancestors and has a straightforward Bayesian interpretation.

Estimation of probabilities of amino acids from observed counts is a very well studied problem and many methods have been proposed [8]. The simplest method to estimate probabilities from counts is the maximum likelihood estimate. Although this method performs well when there is an abundance of data, it is problematic when there is very little data or when the biological sequences' alphabet is large. For this reason, we want to incorporate prior information into the estimation. Intuitively, in instances where we have few observed counts, we want to rely on the prior information more than in instances where we have more observed counts.

The simplest way to incorporate prior information into the probability estimation is to use the pseudocount method. In this method, a vector of "virtual" counts is added to the observed counts of amino acids. Although this method is more robust than the simple maximum likelihood estimate when there is a small amount of data, a single pseudo-count vector cannot encode the relations between symbols in the alphabet such as groupings of amino acids or related protein families. These groupings are important because amino acids tend to appear in groups that share similar biochemical properties such as the hydrophobic group. The presence of one amino acid in the group increases the likelihood of seeing other amino acids from the same group.

A method that addresses these problems is mixtures of Dirichlet distributions [4, 27]. The expectation and the maximum aposteriori value of a (single-component) random variable from the Dirichlet distribution can be viewed as a smoothing process with a pseudo count. A mixture of Dirichlet distributions can encode the grouping information by having a component in the mixture for each group. However, in general it is difficult to compute the optimal Dirichlet components (the pseudo-count vectors) from the data. If there are 10 components in the mixture and there are 20 amino acids, there are about 200 parameters that need to be estimated from the data. The parameters are set by using the EM algorithm to minimize an error function over training data. Since the EM algorithm is subject to local maximum and there are a lot of parameters, it is very difficult to obtain with confidence the best set of components from the data. In the case of amino acids, the estimation is possible and some very good components have been discovered [22]. However, the computation is much more difficult for large alphabets such as in the case of estimating probability distributions over protein families. In the latest version of the Pfam database, there are close to 2,500 protein families [28]. Even with a small number of components, the total number of parameters for Dirichlet mixtures will be very large and will be difficult to optimize.

Another set of methods are based on using substitution matrices [18, 25]. Substitution matrices have the advantages that they explicitly encode the relations between amino acids and can be easily computed even for large alphabets. A problem with substitution matrix-based methods is that each amino acid has a fixed substitution probability with respect to each other amino acid because the matrix is fixed. Heuristic approaches to address these problems use the substitution matrix to set a pseudocount vector [29, 5, 18]. Although these methods perform well in practice, they have little theoretical justification.

One approach to the problem of a fixed substitution matrix is presented by Gribskov and Veretnik [14]. The approach estimates amino acid probabilities by making an assumption that they were derived from a common ancestor and uses a substitution matrix to obtain the probability estimates. Gribskov and Veretnik first computed which of a set of substitution matrices fit the observed counts best using the measure of cross entropy. They typically used a set of 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 PAM matrices. Because they choose one of a set of possible substitution matrices, their model is more flexible.

In this paper we present a mixture model of *common ancestors*. Our approach that builds upon and generalizes the method presented in [14] to allow for efficient exact computation of a richer model. In contrast to previous work on substitution-based methods, our method has a simple Bayesian interpretation and has three advantages over the Gribskov and Veretnik model [14]. The first is that our model is richer since we consider infinitely many

possible matrices instead of 12 choices. Second, we employ a set of priors that can be used to incorporate biological information into the model. Finally, we derive an analytical solution for the weight of each ancestor in our mixture as well as the possible mutation rates. This analytical solution can be computed efficiently in time that depends only on the size of the alphabet.

The mixture of common ancestors method is an exact method for computing the probability distribution over a discrete alphabet given a set of observed counts that takes into account as a prior the relations between elements of the alphabet. The method makes an assumption that the observations all were derived from a common ancestor through mutations. Given this assumption, if the common ancestor is known, as well as the mutation rate, we can easily compute the probability distribution. The observed counts are used to induce a distribution over the possible ancestors and their rate of mutations. Put another way, since apriori, the common ancestor and mutation rate are not known, we "hedge our bets" on the ancestor's identity by maintaining a weighted mixture over the possible ancestors and their mutation rates and show predictions according to the mixture can be computed efficiently.

We present our method in the context of amino acids and later apply the method to compute probability distributions over another larger alphabet (protein families). We present an efficient method for computing the probability estimate. We perform experiments comparing our method versus comparison methods over amino acid counts obtained from an aligned database. We show the ability of the method to handle large alphabets in experiments using this method as a basis for protein family classification and compare the performance to other methods.

2 Preliminaries

Throughout the paper we use the following notation. We denote by Σ the set of possible observations. For instance, in the case of nucleotides $\Sigma = \{A, C, T, G\}$. A sequence of length t of observations is denoted by $x^1, x^2, \ldots, x^t$ where $x^s \in \Sigma$ for $1 \leq s \leq t$. The number of occurrences of $i \in \Sigma$ in $x^1, x^2, \ldots, x^t$ is denoted by n_i^t, that is, $n_i^t = |\{s | x^s = i, 1 \leq s \leq t\}|$. Analogously, the number of observations in $x^1, x^2, \ldots, x^t$ which are *different* from i is denoted by $\tilde{n}_i^t = t - n_i^t$. For convenience we define for all $i \in \Sigma$, $n_i^0 = \tilde{n}_i^0 = 0$.

A major statistical tool in this paper is a mixture model. A mixture model is a combination of simpler models called base or ground models. Each base model induces a probability estimate over events in Σ. Denote the prediction of the jth model on x^s by $P_j(x^s)$. Each base model is associated with a weight denoted w_j. These weights reflect the importance (or reliability) of each model in the mixture. That is, the higher w_j is the more we "trust" the j model. The mixture weights are updated after each observation as follows,

$$w_j^{t+1} = w_j^t \, P_j(x^t) \; . \tag{1}$$

In this formulation the mixture weights are *unnormalized*. The prediction given by the mixture model is the weighted sum of its base model constituents, that is,

$$P(x^t) = \frac{\sum_j w_j^t P_j(x^t)}{\sum_j w_j^t} \; , \tag{2}$$

where j ranges over all base models in the mixture. Note that there is an equivalent formulation in which the mixture weights are being normalized after each prediction while omitting the normalization from Equ. (2).

Equ. (2) is a direct application of Bayes rule assuming that the models are statistically independent. While this assumption is not true in practice, one can still use Eqs. (1) and (2). The formal properties of the Bayes inference algorithm in agnostic settings, i.e., settings in which the independence assumption does not hold, have been studied extensively in information theory, statistics, and learning theory (see for instance instance [16, 15, 26, 30, 11] and the references therein). An attractive property of mixture models that we exploit in this paper is the simple adaptation to new examples via Equ. (1). The weight update procedure is computationally feasible so long as the computation time of the predictions of the base models is reasonable. Furthermore, since the base models can be mixture models themselves and the number of base models may be infinite, as is the case in this paper, a special attention should be devoted to the design of efficient inference and weight update procedures. In order to compute the prediction of the model, we must be able to easily compute Equ. (2).

3 The common ancestor model

The common ancestor model is well motivated in biology and can be described in the context of protein homology. In the protein homology problem, the goal is to determine which proteins are derived from the same common ancestor. The common ancestor model makes the assumption that at some point in the past, each of the protein sequence in a family was derived from a common ancestor sequence. That is, at each amino acid position in the sequence, each observed amino acid occurs because of a mutation (or set of mutations) from a common amino acid ancestor.

Important in estimating the probability distributions over amino acids is determining the common ancestor. Because amino acids have different chemical properties, different common ancestors mutate with different probabilities to different amino acids. We can represent these probabilities in a mutation matrix. A mutation matrix encodes the probabilities that a common ancestor mutates to another amino acid *given that a mutation occurred*. Each row of the mutation matrix corresponds to a common ancestor. Each column of the matrix corresponds to the resulting amino acid. In general we denote an element in the mutation matrix $M_{i,j}$. This element corresponds to the probability of the ith amino acid mutating to the jth amino acid if a mutation occurs. Note that for all i $M_{i,i} = 0$ and $\sum_k M_{i,k} = 1$. In a mutation matrix, the diagonal elements are all zeros because we assume that a mutation occurred.[1] A mutation matrix can easily be derived from a standard substitution matrix as we show below. Using the observed counts we will attempt to determine the common ancestor which determines which row of the matrix to use as the probability distribution.

A second factor in the probability estimation is how likely a mutation is to occur at a given position in an alignment. This depends on the evolutionary distance of the common ancestor. If there is a very short evolutionary distance between the common ancestor and the observed sequence, then there will be very few mutations. However, if the evolutionary distance is very large, there will be a significantly higher number of mutations. The evolutionary distance defines the probability of mutation. In our model, we denote the probability of mutation α. Likewise, the probability that a mutation did not occur is $1-\alpha$.

Assuming that we know the probability of mutation as well as the common ancestor, we can obtain a probability distribution over amino acids. We denote this probability $P_{\alpha,c}$. If we know that the common ancestor is c and the mutation probability is α, the probability of observing an amino acid i is:

$$\forall i \neq c: \quad P_{\alpha,c}(i) = \alpha M_{c,i} \tag{3}$$

$$i = c: \quad P_{\alpha,c}(i) = 1 - \alpha \ . \tag{4}$$

A mutation matrix can be simply obtained from a standard substitution matrix. Let $S_{i,j}$ be the i,j element of a standard substitution matrix S, such as the ones from the BLOSUM of PAM families of substitution matrices in a form where each element is a conditional probability [18, 25]. In our framework, each row in the substitution matrix is a common ancestor with a fixed mutation rate, denoted α_i. With this insight we can now compute the corresponding mutation matrix $M_{i,j}$ as follows. Our framework, described above, implies that $S_{i,i} = 1-\alpha_i$. Furthermore, since each off-diagonal element in the matrix $S_{i,j} = \alpha_i M_{i,j}$ we get the following transformation,

$$M_{i,j} = \begin{cases} \frac{S_{i,j}}{1-S_{i,i}} & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

As we discuss in the next section, the fixed mutation rate, implied by the $S_{i,i}$, is instead replaced with a mixture of potential mutation rates where the mixture weights are automatically estimated by our mixture learning algorithm. Thus, our framework automatically adjusts the weights to the set of probable mutation rates based on the data.

4 Mixtures of common ancestors

The probability distribution over amino acids depends on two factors: the rate of mutation, α, and

[1] Of course there is the possibility that a multiple mutations occurred and then the amino acid mutated back to the original amino acid. We can address this case and the case of multiple mutations by defining mutations and non-mutations in terms of observations. If we observe the common ancestor, then we define that a mutation did not occur even if the common ancestor mutated to another amino acid and mutated back. Similarly, if we observe a different amino acid, we define this as a mutation regardless of the actual number of mutations that occurred from the original common ancestor. Thus the mutation matrix itself gives the probability of observing an amino acid given its common ancestor. This is a reasonable definition, because the mutation matrices are estimated using observed counts.

the common ancestor c. Apriori, we do not know the common ancestor nor the mutation rate. Intuitively, we want to estimate these values using the observed data. We use a Bayesian mixture approach.

Informally, we can estimate the mutation by the spread of the amino acids. If the observed amino acid counts are concentrated on one amino acid, then we can intuitively say that the mutation rate is likely to be low, while if the observed amino acid counts are spread among many amino acids, the mutation rate is likely to be high. Similarly, we can attempt to estimate which amino acid is the common ancestor by the specific amino acids observed.

Formally, we examine the observed counts of each amino acid in the sequence $x^1, \ldots, x^t$. We use the notation of Sec. 2 and denote the counts at time t of a given amino acid i by n_i^t. If the common ancestor is c we can expect the mutation rate to be low if n_c^t is high relative to $\tilde{n}_c^t = \sum_{i \neq c} n_i^t$.

For each common ancestor c, we maintain several possible mutation rates. We denote the set of its possible mutation rates for ancestor c by $R(c)$. For a given mutation $\alpha \in R(c)$, we can use Eqs. (3) and (4) to compute the predictions. We build a mixture model as discussed in Sec. 2: We associate a weight with each possible mutation rate, w_α^t and the prediction of the mixture is the weighted sum of the predictions of each model:

$$P_c(x^t) = \frac{\sum_{\alpha \in R(c)} w_{\alpha,c}^t \, P_{\alpha,c}(x^t)}{\sum_{\alpha \in R(c)} w_{\alpha,c}} \tag{6}$$

The weights are updated based on the performance of the model on the observed data. The models that better predict the observed data will have a higher relative weight in the mixture. For each possible ancestor c and mutation rate $\alpha \in R(c)$, we use the weight update described in Sec. 2, that is,

$$w_{\alpha,c}^{t+1} = w_{\alpha,c} \, P_{\alpha,c}(x^t) \tag{7}$$

In App. A we derive an efficient scheme for computing the mixture over all possible mutation rates the prediction given by Equ. (6). For a common ancestor c, the mixture's predictions, which we get by averaging over all mutation rates, is,

$$x^{t+1} \neq c : P_c(x^{t+1}) \quad = M_{c,i} \frac{\sum_{j \neq c} n_j^t + 1}{\sum_k n_k^t + 2} = M_{c,i} \frac{\tilde{n}_c^t + 1}{t+2} \tag{8}$$

$$x^{t+1} = c : P_c(x^{t+1}) \quad = \frac{n_c^t + 1}{\sum_k n_k^t + 2} = \frac{n_c^t + 1}{t+2} \, . \tag{9}$$

Note that due to our definition of n_i^t the equation above is given in terms of the predictions for x^{t+1}, and not x^t. Also note that our definition of n_i^0 and $\tilde{n}_i^0$ implies that $P_c(x^1) = 1/2 M_{c,x^1}$ when $x^1 \neq c$ and $P_c(x^1) = 1/2$ otherwise. Intuitively, the prior probability, before we see any observation, is such that the probability of mutation happening is equal to the total probability of any of the possible mutation. This prior probability can be modified to reflect prior knowledge, as we describe below.

Eqs. (8) and (9) conform to our intuitions for amino acids. We can see that if we have observed many mutations in the observed amino acid counts, the probability of observing an amino acid other than the common ancestor is high. Likewise, if we observed few mutations in the observed amino acid counts, the probability of observing the common ancestor is high. Thus the spread of the amino acids helps estimate the mutation rate.

We can incorporate some prior biological information into the probability estimate. Notice that the rate of mutation is estimated by the relative number of observed mutations versus observed non-mutations. We can introduce a set of "virtual" observed counts for each common ancestor. These counts behave in a very similar way to pseudo-counts. For ancestor c we define m_c and $\tilde{m}_c$ to be the virtual counts of non-mutations and mutations, respectively. These counts can be used to tune the predictions in Eqs. (8) and (9) to predict more accurately when there are few observed counts. The way these virtual counts are incorporated into the prediction is a simple extension of the derivation of Eqs. (8) and (9) and is also described in App. A. Our final predictions after incorporating the virtual counts are,

$$x^{t+1} \neq c : \quad P_c(x^{t+1}) = \quad M_{c,i} \, \frac{\tilde{m}_c + \tilde{n}_c^t + 1}{m_c + \tilde{m}_c + t + 2} \tag{10}$$

$$x^{t+1} = c : \quad P_c(x^{t+1}) = \quad \frac{m_c + n_c^t + 1}{m_c + \tilde{m}_c + t + 2} \tag{11}$$

Because we do not know apriori the common ancestor, we again apply the mixture technique and evaluate a weighted sum of models each with a different common ancestor. For each common ancestor, c we have a weight at time t, denoted w_c^t. The weight w_c^t reflects the total weight achieved averaging over all possible mutation rates $\alpha \in R(c)$. The initial weight is set to be w_c^1 and using the predictions given by Eqs. (10) and (11), we update the

weights as follows,

$$w_c^{t+1} = w_c^t \, P_c(x^t) \tag{12}$$

The prediction of the mixture at time t, $P(x^t)$, is defined to be:

$$P(x^t) = \frac{\sum_{c \in \Sigma} \, w_c^t \, P_c(x^t)}{\sum_{c \in \Sigma} \, w_c^t} \quad . \tag{13}$$

Since there are only 20 possible candidates for a common ancestor for amino acids ($|\Sigma| = 20$), we can use Eqs. (10) and (11) to compute the prediction of each common ancestor, while taking into account the mixture over the different mutations rates, and obtain the final prediction of the mixture efficiently. Furthermore, we can compute the mixture weights either incrementally (online) using Equ. (12) or, as shown in App. B, we can compute the weights in batch settings using the total counts as follows,

$$w_c^{t+1} = w_c^1 \, \prod_{s=1}^{t} P_c(x^s) \tag{14}$$

$$= w_c^1 \frac{(m_c + n_c^t + 1)! \, (\tilde{m}_c + \tilde{n}_j^t + 1)! \, (m_c + \tilde{m}_c + 2)!}{(m_c + \tilde{m}_c + t + 2)! \, (m_c + 1)! \, (\tilde{m}_c + 1)!} \prod_{l \neq c} M_{c,l}^{n_l^t} \tag{15}$$

Finally, we note that the above derivation can be easily generalized to the case where the pseudo-counts are non-integers. In case of real-valued pseudo-counts the factorial function in Equ. (15) is replaced with the Gamma function. We further discuss this issue in the appendices.

5 Incorporating default models

One advantage of the mixture framework is that we can explicitly incorporate other phenomena into the mixture. That is, the actual probability distribution over amino acids can be dependent on more than just the evolutionary distance and common ancestor. For instance, the probability distribution can depend on the neighboring amino acids or some other structural constraints that are not taken into account in the model presented. Furthermore, in situations when there are vast amounts of data, we would like to "fall back" to a prediction model that is based solely on the empirical counts, namely, the maximum likelihood model.

In order to incorporate a background model B, we add a single component to the mixture along with a smoothing parameter. Specifically, we add a single component Dirichlet distribution with parameter λ whose prediction on x^{t+1} is,

$$P_B(x^{t+1}) = \frac{n_{x^{t+1}}^t + \lambda}{t + |\Sigma|\lambda} \quad . \tag{16}$$

As the number of observations increase, this component will converge to the maximum likelihood estimate. We denote the initial weight of the background predictor by w_B^1. Similar to the weight-update of the ancestor models, the weight-update of the background model is $w_B^{t+1} = w_B^t P_B^t(x^t)$.

Adding the background model as a component to the mixture, we get that the prediction of the mixture (Equ. (13)) becomes,

$$P(x^t) = \frac{w_B^t P_B(x^t) + \sum_c w_c^t P_c(x^t)}{w_B^t + \sum_c w_c^t} \quad . \tag{17}$$

We set the prior weight of this component w_B^1 to be relatively low compared to the priors for the common ancestors, since we would like the background model to have a significant impact on the overall prediction of the mixture only after we have observed non-negligible amounts of data. Thus, initially the prediction of the mixture will be dominated by the predictions of the common ancestor models because of their higher initial weight, but eventually may be dominated by the background component if it systematically performs better than the other components.

6 Using biological information to set the priors

In the common ancestor method, we have several variables that define the prior probabilities. The priors are the initial counts for each common ancestor m_c and $\tilde{m}_c$. We also have the initial weights for each common ancestor w_c^1. Finally, we have the initial weight of the pseudo-count predictor w_B^1 and the initial count λ.

We can use biological information to set reasonable values of many of the priors. We assume we are given a substitution matrix. Typically if the substitution matrix was obtained from a certain family of substitution matrices, there are members of the family which have empirically been shown to perform best. For example, in the BLOSUM family, one

of the most commonly used matrices is the BLOSUM50 matrix. We want to set m_c and $\tilde{m}_c$ for each component so that the default predictions will be the predictions of the BLOSUM50 matrix. This can be done by fixing the ratio between the the number of virtual mutations versus conservations. The magnitude of $m_c + \tilde{m}_c$ defines how much weight the priors are given in the prediction. This can be set based on how much difference in evolutionary distances are observed in the data set.

We set the values of w_i^1 equal to the background probability of amino acid i in the data set. The initial weight of the background model w_B^1 should be set low relative to the weights of the components w_i^1 to allow the method to perform well when there is little data. A reasonable value for λ is $\frac{1}{|\Sigma|}$ or 0.05 in the case of amino acids.

7 Estimating the probabilities of amino acids

To evaluate the method presented in this paper, we measure the performance of the mixtures of common ancestors on estimation of amino acid probabilities and compare this to the performance of the Dirichlet mixtures over the same data set.

A framework for evaluating methods for probability estimation for amino acids is presented in [22]. In this section we follow Karplus' notation and experimental setting for evaluating and comparing our mixture model of ancestors with previously studied models. Karplus presents an information theoretic framework for measuring the effectiveness of a method. Different methods for probability estimation are compared by independently estimating probabilities of columns of multiple alignments from the BLOCKS database [19, 17].

Using the notation from [22], we denote the total count for each amino acid i in a column t by $F_t(i)$. If we use sequence weights such as those presented in [20], $F_t(i)$ are not necessarily integers. The total count for each column is denoted $|F_t| = \sum_i F_t(i)$. Using an entire column, we can estimate the probabilities for each amino acid using the maximum likelihood estimate $\frac{F_t(i)}{|F_t|}$.

In practice, we usually only have access to a small set of observed counts. The goal of the probability estimation method is to obtain estimates over the entire column using only the small set of observed counts. For a small set of observed counts s, we use the method to estimate the probabilities over amino acids $\hat{P}_s(i)$. Intuitively, the better the method, the closer the estimate $\hat{P}_s(i)$ will be to the maximum likelihood estimate over the entire column $\frac{F_t(i)}{|F_t|}$.

A way to measure how close a method estimates this probability is by calculating the encoding cost or conditional entropy. The encoding cost for the sample s in a column t for a given method with estimate $\hat{P}_s(i)$ is given by:

$$H_s(t) = -\sum_i \frac{F_t(i)}{|F_t|} log_2 \hat{P}_s(i) \ . \qquad (18)$$

The more accurate the estimate, the lower the encoding cost. The minimum encoding cost is when the method's estimate equals the maximum likelihood estimate $\hat{P}_s(i) = \frac{F_t(i)}{|F_t|}$. We can measure the performance of a method by computing how much higher the encoding cost is above this minimum. This amount was referred to by Karplus as the *excess entropy*. The excess entropy is called the relative entropy or the Kullback-Leibler in information theory [6]. Let P and Q be two distributions over Σ. Then, the relative entropy between the distributions, denoted $D(Q||P)$, is defined as,

$$D(P||Q) = \sum_{i \in \Sigma} Q(i) \log \left(\frac{Q(i)}{P(i)} \right) \ .$$

In out setting $Q(i) \equiv F_t(i)/|F_t|$ and $P(i) \equiv \hat{P}_s(i)$. Since the focus of this section is comparison to the different probabilistic estimators discussed in [22], we use the notation employed by Karplus.

Karplus examines the expected value of the encoding cost when a sample of size k is chosen. We compute the entropy over all possible samples of size k. This measures the performance of the method after k observations. This gives the following entropy of a column for a given sample size:

$$H_k(t) = \sum_{\text{sample } s, |s|=k} P(s|t) H_s(t) \ , \qquad (19)$$

where $P(s|t)$ is the probability of selecting sample s from column t. By averaging over the entire database we obtain the encoding cost for a method with a given sample size:

$$H_k = \frac{\sum_{\text{column } t} |F_t| H_k(t)}{\sum_{\text{column } t} |F_t|} \qquad (20)$$

$$= -\frac{\sum_{\text{column } t} |F_t| \sum_{\text{sample } s,|s|=k} P(s|t) H_s(t)}{\sum_{\text{column } t} |F_t|}$$

Karplus presents an efficient method for computing the entropy given in equation (20). By comparing encoding costs between two methods, we can compare the performance of two methods.

We perform our experiments over the same data set that was used in [22]. The data consists of sets of observed counts of amino acids taken from the BLOCKS database [19, 17]. The counts are weighted using a position-specific weighting scheme described in [20] with slight variations presented in [27]. The data set was split into disjoint training and test subsets.

For each experiment, we compute equation (20) for a different size of the sample k where $0 \leq k \leq 5$. For each sample size, we compute the excess entropy of the methods over all possible samples drawn from each column. We compare the performance of the mixture of common ancestors (*CA-Mixture*) versus several previously presented methods over the same data set. As baselines for comparison we examine two *zero-offset* methods. These methods are pseudo-count predictors with a fixed initial count for each symbol. We compute the results for a zero-offset method with initial count 1 (*zero-1*) and with a initial count optimized for the dataset $.0481$ (*zero-0.0481*). We also compute the results of a pseudo-count predictor (*pseudo*) with initial counts optimized for the data set and two Dirichlet mixtures. The first Dirichlet mixtures components were obtained from [27] (*Dirichlet-S*) and the second set of components (*Dirichlet-K*) were obtained from [22]. All of the methods optimized with a parameter search for the dataset as described in [22]. This gives the mixture of common ancestors a significant disadvantage because we did not perform a parameter search for the dataset.

The results of these experiments are given in Table 1. As we can see, the mixture of common ancestors performs better than all of the comparison methods except for the Dirichlet mixtures which were optimized for the data. However, the method presented in this paper required very little optimization in order to achieve these results. Also note that the mixture of common ancestors out performs *Dirichlet-S* for small samples. A more complete set of experimental results as well as the mutation matrix and parameter definition files are available at `http://www.cs.columbia.edu/compbio/mca/`.

Sample Size	*CA-Mixture*	*Zero-1*	*Zero-0.0481*	*Pseudo*	*Dirichlet-S*	*Dirichlet-K*
0	0.00633	0.12527	0.12527	0.00610	0.06123	0.00883
1	0.02241	1.07961	0.20482	0.13925	0.05336	0.00115
2	0.05557	1.17080	0.18636	0.13720	0.02402	0.00757
3	0.09525	1.16489	0.16843	0.13097	0.01970	0.01471
4	0.10887	1.13144	0.15311	0.12350	0.02083	0.02740
5	0.11337	1.09164	0.14203	0.11804	0.02455	0.03943

Table 1: Excess entropy for different probability estimation methods under different sample sizes over BLOCKS database. The encoding costs were computed over the BLOCKS database for CA-Mixture, Zero-1, Zero-0.0481, and Pseudo and were consistent with previously published results. The encoding costs for Dirichlet-S and Dirichlet-K reported are published results [22]. Note that the Dirichlet mixtures were obtained with a parameter search in order to minimize entropy over the dataset, while the mixture of common ancestors results are without parameter optimization.

8 Experiments with protein families

As we have shown, the method presented is comparable in performance to other probability estimation methods over amino acids. However, the main advantage to the mixture of common ancestors is that it can handle large alphabets. We apply the mixture of common ancestors presented in this paper to the estimation of probabilities over protein families.

In the current version of the Pfam database, there are approximately 2,500 protein families [28]. A Dirichlet mixture over this alphabet would contain many parameters making it difficult to optimize the parameters over the data. If we assume that with a 2,500 symbol alphabet, there are 100 components, this corresponds to 250,000 parameters. The mixture of common ancestors, however, does not require this kind of training because the necessary parameters can easily be computed directly from the data.

Assuming that we have many sets of observed counts of Σ we can derive a matrix and prior weights from the data. To derive a mutation matrix we use the method presented in [18] to derive a substitution matrix and then convert it to a mutation matrix using the method presented in Sec. 3. We can set the prior component weights w_i^1 to the background probabilities computed over the data.

We apply the mixture of common ancestors to estimating distributions over protein families. We are interested in estimating probability distributions from short subsequences of amino acids that are

contained in the protein family and using these distributions for protein classification [3, 1]. Probability distributions over protein families conditional on these short subsequences are used for protein family classification using sparse Markov transducers (SMTs) [10]. Because SMTs are beyond the scope of this paper, we present a very simple model for protein family classification using subsequences and show how the method in this paper significantly improves the accuracy of protein classification.

In this classification model, we view a protein sequence as a set of the subsequences of amino acids that it contains. These subsequences are obtained by a sliding window. For these experiments, we use subsequences of fixed size 6. For each subsequence, we compute a probability distribution over protein families. This probability corresponds to how likely the subsequence is to have originated from that family. These probabilities are computed from a database of classified protein sequences. A protein is classified into a family by computing the probability distribution over protein families for each subsequences. For each family, we compute a score for a protein by computing the normalized sum of logs of the probabilities of the subsequences in the protein originating from that family. This corresponds to assuming that the subsequences are conditionally independent. The protein is classified by determining which family has the highest score.

The key to classifying proteins under this model is the estimation of the probability distribution over protein families for each subsequence. We estimate the probabilities from the observed counts of the subsequence in the database. We compare the mixture of common ancestors to a simple pseudo-count. As we discussed earlier, because of the large alphabet size, it is difficult to use Dirichlet mixtures on this data.

To create the mixture of common ancestors, we first create a data set from the Pfam database version 5.5. In this database, there are a total of 2478 protein families. Our data set is created by obtaining all sequences of length 6 present in the database. For each sequence we obtain counts of how many proteins in each family contain that sequence. This gives us sets of counts of over protein families. There are a total of $13,252,465$ distinct sequences of length 6 amino acids giving a total of $13,252,465$ sets of counts. Using this data we create a mutation matrix and set the prior weights as described above. The mutation matrix is of size 2478×2478.

Family	Mixture of Common Ancestors	Pseudo-counts
14-3-3	.54	.43
2-Hacid_DH	.90	.82
2-oxoacid_dh	.89	.89
3_5_exonuclease	.64	.34
3A	.67	.68
3Beta_HSD	.73	.54
3HCDH	.94	.56
4A_glucanotrans	.76	.56
4HPPD_C	.65	.54
5-FTHF_cyc-lig	.76	.65

Table 2: ROC_{50} scores showing results of protein classification over the Pfam database using mixture of common ancestors and pseudo-count predictors.

We split each family in the Pfam database into a training and testing portion by a ratio of $4:1$. There are a total of $228,984$ protein sequences in the training set and $56,029$ sequences in the testing set. Over the training portion, we compute the probabilities over protein families associated with each sequence of 6 amino acids obtained by a sliding window over the proteins. Using these estimates we classify a protein from the test set as follows. For each protein we extract the length 6 amino acids from the protein using a sliding window. For each protein family, we compute a score which is the length normalized product of the probabilities of the subsequences predicting that family. We classify a protein into the family with the highest score. For comparison, we estimate the probabilities using both the mixture of common ancestors and the pseudo-count method with a virtual count of $\frac{1}{2,478}$.

To evaluate the protein classification we compute the ROC_{50} score [13]. The ROC_{50} score compute the normalized area under the curve that plots true positives versus false positives up to 50 false positives. Table 2 compares the ROC_{50} scores for the first 10 protein families in the Pfam database using the two methods. The results for the complete Pfam database as well as the protein family mutation matrix and parameter definition files are available at `http://www.cs.columbia.edu/compbio/mca/`.

9 Discussion

We have presented the mixture of common ancestors and applied it to estimating amino acid prob-

abilities and protein family probabilities from observed counts. The method is effective for large alphabets. The mixture of common ancestors leverages well-understood techniques for estimating substitution matrices from data [18, 25]. Using a variant of these matrices, *mutation matrices*, the method explicitly encodes the relationships between the symbols of the alphabet.

The mixture of common ancestors has several advantages over previous methods. Unlike previous substitution matrix-based methods, the mixture of common ancestors has a Bayesian interpretation. The method also performs well with both few and many observations. Unlike Dirichlet mixtures, the mixture of common ancestors parameters can be easily computed from training data. This give the advantage of being able to handle large alphabets such as protein families.

In addition, the method is strongly biologically motivated. The method builds on the model presented by Gribskov and Veretnik which can be described in our context. [14]. The Gribskov and Veretnik model used a common ancestor model with a notion of evolutionary distance. They first computed which of a set of matrices fit the observed counts best using cross entropy which in our framework, is equivalent to choosing an evolutionary distance. Once they choose a matrix, they apply a mixture over the common ancestor (rows) of the matrix. Our method has three advantages. The first is that our model is richer because the evolutionary distance is continuous versus one of 12 choices specified by the choice of matrices. Second, we have a rich set of priors that can be used to incorporate biological information into the predictor. Finally, we have an efficient analytical solution to the prediction over all possible evolutionary distances versus the expensive computation in their method.

Future work involves investigating what kinds of mutation matrices and which prior weights lead to optimal performance under different alphabets. One direction of future work is to compute mutation matrices directly from the data optimized for this model as opposed to computing them from known substitution matrices.

Appendices

A Computing the prediction of a single ancestor model

In this section we describe the derivation of an efficient procedure for computing the prediction of the mixture over mutation rate for a common ancestor c. We first discuss the case of a finite set of mutation rates and then derive a closed form at the limit of infinite rates. The second part of our analysis is uses a specific form of the Dirichlet distribution (see for instance [7]) and can be skipped if the reader is familiar with Bayesian inference using a conjugate family of distributions.

Given a common ancestor c, we wish to estimate a probability distribution over amino acids using the the set of observed counts n_i^t. We do not know the mutation probability α_c for the component. We thus would use the observed number of mutations and non-mutations to help determine what the mutation rate it and build a mixture of the possible mutation rates as discuss in Sec. 2. Let $R(c)$ be a finite set of possible mutation rates for c. We first assume that $R(c)$ is composed of evenly spaced rates in $(0, 1)$. Formally, let $|R(c)| = n$ then $R(c) = \{i/n | 1 \leq i \leq n-1\}$. Each $\alpha \in R(c)$ constitute a component in the mixture. The weight of a α after t observations $(x^1, \ldots, x^t)$ is denoted by w_α^t. Its initial weight, w_α^1, is set to a predefined value and is called a prior weight.

Each possible rate $\alpha \in R(c)$ induces a probability distribution over Σ as given by Eqs. (3) and (4). To remind the reader, after each observation x^t, the weight is updated according to the component's prediction, namely $w_\alpha^{t+1} = w_\alpha^t P_{\alpha,c}(x^t)$. Unraveling this recursive weight update we get that the weight of the component corresponding to α after t observations is

$$\begin{aligned} w_\alpha^{t+1} &= w_\alpha^1 \prod_{s=1}^{t} P_{\alpha,c}(x^s) \\ &= w_\alpha^1 (1-\alpha)^{n_c^t} \prod_{j \neq c} (\alpha M_{c,j})^{n_j^t} \end{aligned} \quad (21)$$

The prediction of the entire mixture is obtained by computing weighted average of the predictions normalized by the total sum of the weights,

$$\begin{aligned} P_c(x^{t+1}) &= \frac{\sum_{\alpha \in R(c)} w_\alpha^{t+1} P_{\alpha,c}(x^{t+1})}{\sum_{\alpha \in R(c)} w_\alpha^{t+1}} \\ &= \frac{\sum_{\alpha \in R(c)} w_\alpha^1 (1-\alpha)^{n_c^t} \prod_{j \neq c} (\alpha M_{c,j})^{n_j^t} P_{\alpha,c}(x^{t+1})}{\sum_{\alpha \in R(c)} w_\alpha^1 (1-\alpha)^{n_c^t} \prod_{j \neq c} (\alpha M_{c,j})^{n_j^t}} \end{aligned} \quad (22)$$

Using the definition of $\tilde{n}_j^t$ we rewrite $P_c(x^{t+1})$ as follows,

$$P_c(x^{t+1}) =$$

$$= \frac{\left(\prod_{j\neq c} M_{c,j}^{n_j^t}\right) \sum_{\alpha\in R(c)} (1-\alpha)^{n_c^t}\, \alpha^{\sum_{j\neq c} n_j^t}\, P_{\alpha,c}(x^{t+1})\, w_\alpha^1}{\left(\prod_{j\neq c} M_{c,j}^{n_j^t}\right) \sum_{\alpha\in R(c)} (1-\alpha)^{n_c^t}\, \alpha^{\sum_{j\neq c} n_j^t}\, w_\alpha^1}$$
$$= \frac{\sum_{\alpha\in R(c)} (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t}\, P_{\alpha,c}(x^{t+1})\, w_\alpha^1}{\sum_{\alpha\in R(c)} (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t}\, w_\alpha^1} \quad . \tag{23}$$

Equ. (23) can be computed efficiently if $|R(c)|$ is finite and relatively small. If we would like to work with continuously many values for α we replace the summations with the integrals and the prior weights w_α^1 with a *prior* distribution $\mu(\alpha)$ and get,

$$P_c(x^{t+1}) \;=\; \frac{\int_0^1 P_{\alpha,c}(x^{t+1})\, (1-\alpha)^{n_c^t}\, \alpha^{\bar{n}_c^t}\, \mu(\alpha)\, d\alpha}{\int_0^1 (1-\alpha)^{n_c^t}\, \alpha^{\bar{n}_c^t}\, \mu(\alpha)\, d\alpha} \tag{24}$$

We now need to consider two cases depending whether $x^{t+1} \neq c$ or $x^{t+1} = c$. In the first case we get

$$x^{t+1}\neq c: \quad P_c(x^{t+1}) = \frac{\int_0^1 \alpha M_{c,x^{t+1}}\, (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t}\, \mu(\alpha)\, d\alpha}{\int_0^1 (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t}\, \mu(\alpha)\, d\alpha}$$
$$= \frac{\int_0^1 M_{c,x^{t+1}}\, (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t+1}\, \mu(\alpha)\, d\alpha}{\int_0^1 (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t}\, \mu(\alpha)\, d\alpha} \tag{25}$$

and in the second case we get

$$x^{t+1}= c: \quad P_c(x^{t+1}) = \frac{\int_0^1 (1-\alpha)\, (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t}\, \mu(\alpha)\, d\alpha}{\int_0^1 (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t}\, \mu(\alpha)\, d\alpha}$$
$$= \frac{\int_0^1 (1-\alpha)^{n_c^t+1}\, \alpha^{\tilde{n}_c^t}\, \mu(\alpha)\, d\alpha}{\int_0^1 (1-\alpha)^{n_c^t}\, \alpha^{\tilde{n}_c^t}\, \mu(\alpha)\, d\alpha} \quad . \tag{26}$$

To further simplify Eqs. (25) and (26) we need to choose a specific density function for $\mu(\alpha)$. Let us first analyze a simple case where $\mu(\alpha)$ is the uniform distribution over $[0,1]$. We use the identity,

$$\int_0^1 x^m (1-x)^n dx = \frac{m!\, n!}{(m+n+1)!} \quad . \tag{27}$$

Note also that from definition we get that $\tilde{n}_c^t + n_c^t = t$. Then, Eqs. (25) and (26) simplify to,

$$\begin{aligned} x^{t+1}\neq c: \quad P_c(x^{t+1}) &= M_{c,x^{t+1}}\, \frac{n_c^t!\, (\tilde{n}_c^t+1)!}{(n_c^t+\tilde{n}_c^t+1+1)!}\, \frac{(n_c^t+\tilde{n}_c^t+1)!}{n_c^t!\, \tilde{n}_c^t!} \\ &= M_{c,x^{t+1}}\, \frac{\tilde{n}_c^t+1}{n_c^t+\tilde{n}_c^t+2} \\ &= M_{c,x^{t+1}}\, \frac{\tilde{n}_c^t+1}{t+2} \quad , \end{aligned} \tag{28}$$

$$\begin{aligned} x^{t+1}= c: \quad P_c(x^{t+1}) &= \frac{(n_c^t+1)!\, \tilde{n}_c^t!}{(n_c^t+\tilde{n}_c^t+1+1)!}\, \frac{(n_c^t+\tilde{n}_c^t+1)!}{n_c^t!\, \tilde{n}_c^t!} \\ &= \frac{n_c^t+1}{n_c^t+\tilde{n}_c^t+2} \\ &= \frac{n_c^t+1}{t+2} \quad . \end{aligned} \tag{29}$$

In the general case where $\mu(\alpha)$ is not the uniform distribution, we can still compute efficiently the integrals over α analytically in some cases. One particularly convenient prior distribution for α is the Dirichlet distribution [7], which also has good formal properties. In our setting the specific Dirichlet distribution employs two hyper-parameters which we denote by m_c and $\tilde{m}_c$ and the form of $\mu(\alpha)$ becomes,

$$\mu(\alpha) \propto (1-\alpha)^{m_c} \alpha^{\tilde{m}_c} \quad .$$

These hyper-parameters can viewed as an additional set of pseudo-counts which biases the mixture's prediction. The larger these pseudo-counts are the more we rely on the prior knowledge. If m_c and $\tilde{m}_c$ are integers, then Eqs. (25) and (26) now become,

$$x^{t+1}\neq c: \quad P_c(x^{t+1}) = \frac{\int_0^1 M_{c,x^{t+1}}\, (1-\alpha)^{m_c+n_c^t}\, \alpha^{\tilde{m}_c+\tilde{n}_c^t+1}\, d\alpha}{\int_0^1 (1-\alpha)^{m_c+n_c^t}\, \alpha^{\tilde{m}_c+\tilde{n}_c^t}\, d\alpha} \tag{30}$$

$$x^{t+1}= c: \quad P_c(x^{t+1}) = \frac{\int_0^1 (1-\alpha)^{m_c+n_c^t+1}\, \alpha^{\tilde{m}_c+\tilde{n}_c^t}\, d\alpha}{\int_0^1 (1-\alpha)^{m_c+n_c^t}\, \alpha^{\tilde{m}_c+\tilde{n}_c^t}\, d\alpha} \quad . \tag{31}$$

and using Equ. (27) again, we get

$$x^{t+1}\neq c: \quad P_c(x^{t+1}) = M_{c,x^{t+1}}\, \frac{\tilde{m}_c+\tilde{n}_c^t+1}{\tilde{m}_c+m_c+t+2} \tag{32}$$

$$x^{t+1}= c: \quad P_c(x^{t+1}) = \frac{m_c+n_c^t+1}{\tilde{m}_c+m_c+t+2} \tag{33}$$

In the more general case when the pseudo-counts are real-valued we need replace the factorial function with the Gamma function. Nonetheless, the final form of prediction is remains the same and it is still given by Eqs. (32) and (33). We omit the details of the derivation since it is a straightforward generalization of the derivation above using the properties of the Gamma function which are described in [7].

B Batch computation of the mixture weights

The mixture weight for a given common ancestor c is

$$w_c^{t+1} = w_c^1 \prod_{i=1}^{t} P_c(x^i) \quad , \tag{34}$$

with $P_c(x^i)$ defined in Eqs. (32) and (33). We want to show that

$$w_c^{t+1} = w_c^1 \frac{(m_c+n_c^t+1)!\, (\tilde{m}_c+\tilde{n}_c^t+2)!\, (m_c+\tilde{m}_c+2)!}{(m_c+\tilde{m}_c+t+2)!\, (m_c+1)!\, (\tilde{m}_c+1)!} \prod_{l\neq c} M_{c,l}^{n_l^t} \quad . \tag{35}$$

Since the weight w_c^{t+1} is a product of the predictions, it is invariant to the ordering of the observations. We thus can reorder the observations as follows. Let $x^1, ..., x^{n_c^t}$ be all of the observations of the non-mutations (i.e., for $1 \leq s \leq n_c^t,\ x^i = c$) and let $x^{n_c^t+1}, ..., x^t$ be the remaining observations. We can also factor out the $M_{c,i}$ terms. In this ordering the mixture weight is then

$$w_c^{t+1} = w_c^1 \prod_{i=1}^{n_c^t} \frac{m_c + i + 1}{m_c + \tilde{m}_c + i + 2} \prod_{j=n_c^t+1}^{t} \frac{\tilde{m}_c + j - n_c^t + 1}{m_c + \tilde{m}_c + j + 2} \prod_{\neq c} M_{c,l}^{n_l^t}$$

$$= w_c^1 \frac{(m_c + n_c^t + 1)!(\tilde{m}_c + \sum_k n_k^t - n_c^t + 1)!(\tilde{m}_c + m_c + 2)!}{(m_c + 1)!(\tilde{m}_c + 1)!(m_c + \tilde{m}_c + \sum_k n_k^t + 2)!} \prod_{l \neq c} M_{c,l}^{n_l^t}$$

$$= w_c^1 \frac{(m_c + n_c^t + 1)!}{(m_c + 1)!} \frac{(\tilde{m}_c + \tilde{n}_c^t + 1)!}{(\tilde{m}_c + 1)!} \frac{(\tilde{m}_c + m_c + 2)!}{(m_c + \tilde{m}_c + t + 2)!} \prod_{l \neq c} M_{c,l}^{n_l^t}$$

which is Equ. (35). Similar derivation is used with the Gamma function when m_c and $\tilde{m}_c$ are real-valued.

References

[1] A. Apostolico and G. Bejerano. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. In *Proceedings of RECOMB2000*, 2000.

[2] P. Baldi, Y. Chauvin, T. Hunkapiller, and M. A. McClure. Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences of the United States of America*, 91(3):1059–1063, 1994.

[3] G. Bejerano and G. Yona. Modeling protein families using probabilistic suffix trees. In *Proceedings of RECOMB99*, pages 15–24. ACM, 1999.

[4] M. Brown, R. Hughey, A. Krogh, I. Mian, K. Sjolander, and D. Haussler. Using Dirichlet mixture priors to derive hidden Markov models for protein families. In C. Rawlings, editor, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 47–55. AAAI Press, 1995.

[5] J. M. Claverie. Some useful statistical properties of position-weight matrices. *Computers and Chemistry*, 18:287–294, 1994.

[6] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[7] M. H. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, New York, 1970.

[8] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge UP, 1998.

[9] S. R. Eddy. Multiple alignment using hidden Markov models. In C. Rawlings, editor, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 114–120. AAAI Press, 1995.

[10] Eleazar Eskin, William Noble Grundy, and Yoram Singer. Protein family classification using sparse markov transducers. In *Proceedings of the Eigth International Conference on Intelligent Systems for Molecular Biology*, Menlo Park, CA, 2000. AAAI Press.

[11] Yoav Freund. Predicting a binary sequence almost as well as the optimal biased coin. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, 1996.

[12] M. Gribskov, R. Lüthy, and D. Eisenberg. Profile analysis. *Methods in Enzymology*, 183:146–159, 1990.

[13] M. Gribskov and N. L. Robinson. Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers and Chemistry*, 20(1):25–33, 1996.

[14] M. Gribskov and S. Veretnik. Identification of sequence patterns with profile analysis. *Methods in Enzymology*, 266:198–212, 1996.

[15] David Haussler and Manfred Opper. Mutual information, metric entropy, and cumulative relative entropy risk. *Annals of Statistics*, 25(6), December 1997.

[16] David Haussler and Manfred Opper. *The Mathematics of Information Coding, Extraction and Distribution*, chapter Worst case prediction over sequences under log loss. Springer Verlag, 1998.

[17] J. G. Henikoff and S. Henikoff. Blocks database and its applications. . *Methods in Enzymology*, 266, 1996.

[18] J. G. Henikoff and S. Henikoff. Using substitution probabilities to improve position-specific scoring matrices. *Computer Applications in the Biosciences*, 12:135–143, 1996.

[19] S. Henikoff and J. G. Henikoff. Automated assembly of protein blocks for database searching. *Nucleic Acids Research*, 19(23):6565–6572, 1991.

[20] S. Henikoff and J. G. Henikoff. Position-based sequence weights. *Journal of Molecular Biology*, 243:574–578, 1994.

[21] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 1999. To appear.

[22] Kevin Karplus. Regularizers for estimating distributions of amino acids from small samples. *Technical Report UCSC-CRL-95-11*, 1995.

[23] Kevin Karplus, Christian Barrett, and Richard Hughey. Hidden markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856, 1998.

[24] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[25] R. M. Schwartz and M. O. Dayhoff. *Atlas of Protein Sequence and Structure*, chapter Matrices for detecting distant relationships, pages 353–358. National Biomedical Research Foundation, Silver Spring, MD, 1978.

[26] Y. M. Shtar'kov. Universal sequential coding of single messages. *Problems of information Transmission (translated from Russian)*, 23:175–186, July-September 1987.

[27] K. Sjolander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. S. Mian, and D. Haussler. Dirichlet mixtures: A method for improving detection of weak but significant protein sequence homology. *Computer Applications in the Biosciences*, 12(4):327–345, 1996.

[28] E. Sonnhammer, S. Eddy, and R. Durbin. Pfam: a comprehensive database of protein domain families based on seed alignments. *Proteins*, 28(3):405–420, 1997.

[29] R. L. Tatusov, S. F. Altschul, and E. V. Koonin. Detection of conserved segments in proteins: iterative scanning of sequence databases with alignment blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 91(25):12091–12095, 1994.

[30] F.M.J. Willems, Y.M. Shtarkov, and T.J. Tjalkens. The context tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.